بسم الله الرحمن الرحيم

Sudan University of Science and Technology

Collage of Graduate Studies

Collage of Computer Science and Information Technology

Enhancing Compression Ratio of Variable-Byte Scheme for Inverted File Index

تحسين معدل الضغط لطريقة الثمانية المتغيرة لمفهرس الملف المعكوس لخوارزمية

Variable-Byte

A Thesis Submitted in Partial Fulfillment of the requirements of M.Sc. in computer science

By

Aljaily Mamoun Hassan Altybe

Supervisor:

Dr. Eng. Faiz Yousif Mohammed Yousif

July, 2017

DEDICATION:

This thesis is dedicated to my mother and my father; it is also dedicated to my brothers and sisters and my colleagues and friends.
And to each of the lit up with his knowledge other's mind or solve the correct answer puzzled questioners he showed morals humility scientists and the beauty of those who know.

ACKNOWLEDGEMENT:

Praise to Allah, Lord of the Worlds, peace and prayers be on human teacher Prophet Muhammad and his family and friends and who had followed them in a good way to the last day of the human's life.

My thanks to all who contributed in the output of this research, and to all those who were the cause of education and guiding and helping me.

And many thanks to supervisors Dr. Faiz Yousif Mohammed Yousif who make full effort in guiding and steering while working in the research.

May Allah reward you and guide your steps.

ABSTRACT:

One of the most important characteristics of file compression is to preserve memory with all its temporary and permanent images, in addition to maintaining the speed of performance in the process of compression and decompression and increase the frequency range in the transmission and reception from long distances.

The examined the performance of Variable-Byte compression scheme over sequences of integers. Variable-Byte algorithm which has high compression speed and decompression speed while the algorithm's compression ratio is weak in the case of small integers.

Has been designed enhanced compression algorithm (Variable-5bits) based on Variable-Byte algorithm, evaluated the performance of Variable-5bits thorough experiments on synthetic data sets with random integers in the range 0 – 2^^{31} ,the ratio of compression is improved from 4 to 6.4 in the case of small integers. In the case of large integers, the variable-5bits algorithm is roughly equal in the compression ratio with the Variable-Byte algorithm.

A number of experiments were performed to test the compression ratio in the variable-5bits in different and varied samples when the range is small between integers, the compression ratio is high and when the range is large between the integers we get a low compression ratio.

المستخلص :

من أهم خصائص ضغط الملفات الحفاظ علي الذاكرة بكافة صورها المؤقته والدائمة، بالإضافه إلي الحفاظ علي سرعة الآداء في عملية الضغط وفك الضغط ويزيد من نطاق التردد في عمليات الإرسال والإستقبال من مسافات بعيدة .

تم دراسة آداء الضغط لخوارزمية Variable-Byte علي سلسة الأعداد الصحيحة فوجدت أن سرعة الضغط وفك الضغط لديها عالية بينما نسبة أو معامل ضغط الخوارزمية ضعيفه في حالة الأعداد الصحيحة الصغيرة.

تم تصميم خوارزمية ضغط مُحَسِنَه (Variable-5bits) لخوارزمية Variable-Byte ، تم تقييم الآداء للخوارزمية المحسنه من خلال إجراء التجارب على مجموعة بيانات تجميعية مولدة عشوائياً في مدى الأعداد الصحيحة من 0 - 2^{31} ، تحسن فيها معامل الضغط من 4 – 6.4 في حالة الأعداد الصحيحه الصغيرة ، وفي حالة الأعداد الصحيحة الكبيرة نجد أن خوارزمية Variable-5bits متساوية تقريباً في معامل الضغط مع خوارزمية Variable-Byte .

أجريت عدد من التجارب لإختبار معامل الضغط في الخوارزمية المحسنه في عينات مختلفة ومتنوعة فعندما يكون المدى صغير بين الأعداد الصحيحة يكون معامل الضغط عالي وعندما يكون المدى كبير بين الأعداد الصحيحة نحصل علي معامل ضغط منخفض .

Table of Contents:

LIST OF FIGURES:

CHAPTER ONE

INTRODUCTION

1- INTRODUCTION

1.1- Introduction:

Information retrieval systems deal with a huge amount of data that needs to be organized as inverted index [1]. The use of inverted indexes and compression techniques is partially accountable for the current performance achievement of web search engines [2]. Large amount of data their performance becomes constrained by the speed at which data can be read or written and need it more storage to store. Compression reduces the physical size of the inverted index, saving storage cost and improves the performance [3], [4]. Keeping data in a compressed format can preserve memory and allow the transfer of data to and from memory in a shorter time with lower processing cost and enhance I/O bandwidth. Fast compression can reduce query response times [5].

While the performance of an information retrieval (IR) system can be enhanced through the compression of its posting lists, proposed many compression techniques deal with different types of posting information (document ids, frequencies, positions) [6].

There are some factors which influence compression performance like the number of integers and range of this integer in index.

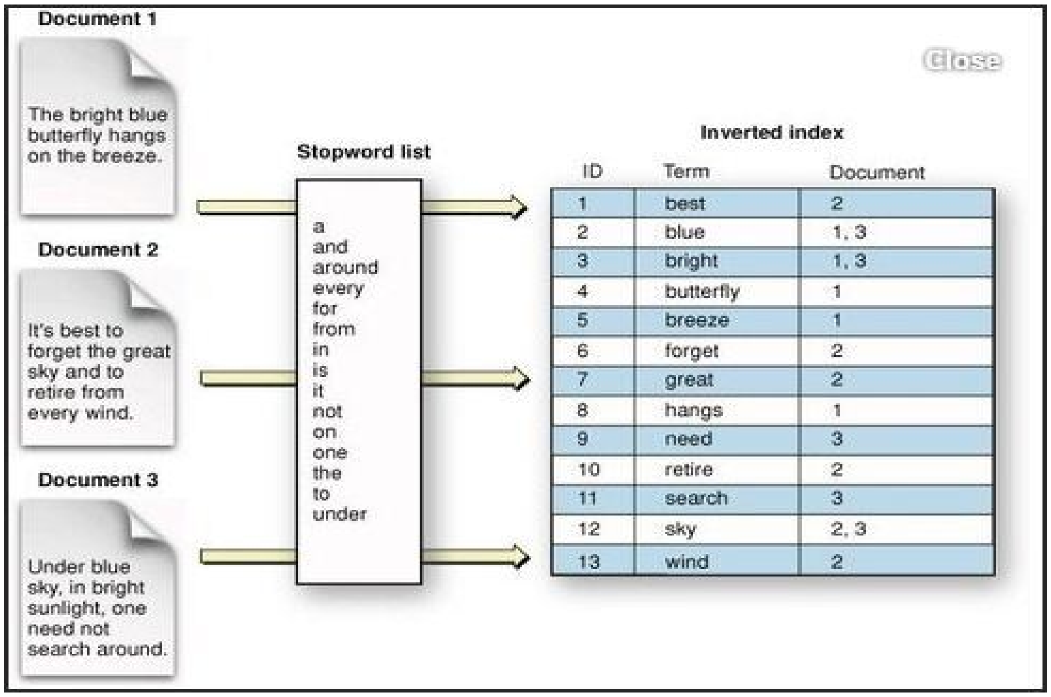
In our study, we focus on the performance of compression applied over sequence of integers. Indexes are mapped to such sequences of integers.

1.2- Inverted Index Structure:

An inverted index is a data structure that stores a list of distinct terms which are found in the collection, this list is called a dictionary, lexicon or a term index. For each term a list of all documents that contain this term is attached, and it is known as the posting list.

For each unique indexed term, the inverted index contains a posting list, where each posting contains the occurrences information (e.g. frequencies, and positions) for documents that contain the term [7].
Inverted index construction is done by collecting the documents that form the corpus.
Afterwards the preprocessing operation is done on the documents to obtain the vocabulary terms; this term is used to build the forward index (document-term index) by creating a list of the words that are in each document [8].

Postings in inverted lists are usually ordered by increasing **See Fig 1.1.**



Document 1: The bright blue butterfly hangs on the breeze.

Document 2: It's best to forget the great sky and to retire from every wind.

Document 3: Under blue sky, in bright sunlight, one need not search around.

ID	Term	Document
1	best	2
2	blue	1, 3
3	bright	1, 3
4	butterfly	1
5	breeze	1
6	forget	2
7	great	2
8	hangs	1
9	need	3
10	retire	2
11	search	3
12	sky	2, 3
13	wind	2

Fig 1.1 Inverted Index

1.3- Techniques for Inverted List Compression

An inverted index is composed of inverted lists, each one being an ordered list of integers. The main idea of index compression is to encode the list of integers using as few bits as possible. Instead of storing the raw integer in a 32 bit machine word, the goal is to store each integer using the smallest number of bits possible **See Fig 1.2.**

For each term *t*, we store a list of all documents that contain *t*.

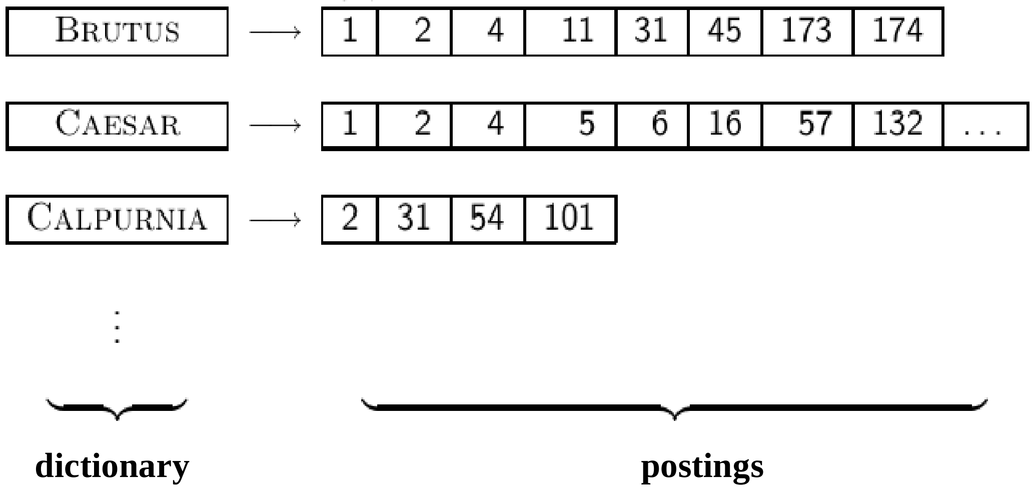


Fig 1.2 Construct the Inverted Index

1.4- Problem statement:

Many applications, such as search engines and information retrieval systems, deal with large amounts of data and these data are often stored in the form of arrays of integers. The large amount of database, their store required more space, and performance becomes constrained by the speed at which data can be sent, read or written.

Variable-Byte algorithm is used to compression the integers and known faster than traditional bit oriented methods, the main drawback of this algorithm inefficient in term of compression ratio.

1.5- Research Question:

How we can enhance the compression ratio of Variable-Byte scheme?

1.6- Objective of the Research:

The goal of this research is to enhance VByte scheme with respect to compression ratio in inverted index.

The study should meet the following objectives:

- Examining and comparing the performance of proposed scheme with respect to compression ratio and speed of compression and decompression.
- Evaluate the effect of different factors such as integer variant, fixed number of integers, variable number of integers.

1.7- Research Scope:

The scope of this research is in the Information Retrieval area. Within the field of information retrieval we focus on compression integers of inverted index.

1.8- Research Organization:

The present research is organized into five chapters entitled: introduction; background and related work; analysis and design; experiment and discussion the results and finally conclusions and future work.

Chapter One of the research is mainly an introduction to the research which, problem statement and the aims of the research, in addition to the scope of the research, and finally an organization of the chapters.

Chapter Two include compression definition and type of compression, background relating to the research. The background gives an overview of compression used in inverted index of information retrieval (IR). It is then followed by the related works.

Chapter Three include explain the old method of compression algorithm and then show weaknesses. After this we will explain the new algorithm (improving the old algorithm).

Chapter Four include generating synthetic data and discussion and evaluation the results.

Chapter Five include conclusion and future work.

1.9- Summary:

In this chapter we introduction to the research and explain problem statement and the aims of the research, in addition to the scope of the research, and finally an organization of the chapters.

In the next chapter we definition the compression and type of compression, background relating to the research and related work.

CHAPTER TWO

BACKGROUND & RELATED WORK

2. BACKGROUND

2.1- Introduction:

In this chapter, we describe the basic concepts that are required to conduct this research. We first describe the basic concepts about compression, second, we show Comparison Characteristics (factors), third, we show compression techniques, Final, for related works.

2.2- Compression Definition:

In signal processing, data compression, source coding, or bit-rate reduction involves encoding information using fewer bits than the original representation.

There are some factors which influence compression performance like the distribution of integers and different between it.

2.3- Light-weight Compression:

Poor storage performance of disk and memory storage becomes a limiting factor for many applications such as information retrieval systems. Compression will allow storing more data in cache during query processing, which can result in faster operation. Compression techniques must have both high compression ratio and decompression speed to enhance the performance of information retrievals. Also, lossless compression is needed for most IR operations. Indeed, the compression scheme should permit recovery of the original data from its compressed form. To achieve this, light-weight compression methods are used. These methods aim to compress while minimizing CPU usage.

Some of the most common light-weight compression methods over include Binary Packing/Frame Of Reference (FOR), Delta coding, Variable-Byte Coding, Simple9, and Simple16. These coding techniques are often used in inverted index compression for IR.

Recently, patched versions of FOR and Delta coding known as PFOR and PForDelta have been used for inverted list compression. Patched compression schemes use the super-scalar facility of a modern CPU._As a result, these techniques provide aggressive compression and decompression speed as well as a modest compression ratio.

This research aims to enhance one of these techniques that (Variable-Byte) with respect to compression ratio and decompression speed in inverted index.

2.4- Types of Compression:

According to the characteristics of compressed data and data recovery from compressed form, compression techniques can be divided into two categories.

2.4.1- Lossy Compression:

In lossy compression schemes, some information has been discarded while compressing the original data. The main objective of this approach is to reduce the size by keeping the overall structure of the file and eliminating some detail or redundant or unnecessary information of the original data. Therefore, it generates an approximation of original data in exchange for a smaller size. Lossy compression techniques are generally used in audio, video or image compression.

2.4.2- Lossless Compression:

Lossless compression allows recovering the original data from its compressed form without any information loss. Lossless compression reduces the size as well as preserving all information of the original file. Lossless compression techniques are often used with text documents.

2.5 - Comparison Characteristics:

Comparing the compression schemes is not an easy task. There are many characteristics including compression ratio, compression speed and decompression speed. In order to get better performance, we need to preserve disk and also minimize processing so, we evaluate enhanced compression scheme mainly by compression ratio, compression speed and decompression speed.

2.6- Compression Techniques

Compression can be done on various types of posting list like document ids, frequencies, positions. In our study, we mainly concerned with compression of document ids.

In recent years, several compression schemes have been developed which have fast decompression speed with moderate compression ratio. Our goal is to achieve significant benefit from compression in inverted index.

We can classify integer compression into two primary types depending on the nature of the output of encoding: schemes that use a variable number of bytes or bits to compress every integer and those that use a fixed number of bytes or bits to compress a variable number of integers.

Variable Length Output:

It takes a fixed input length such as a single integer and processes it into variable length output depending on the input value. This can be classified into three sub-categories:

1- Byte-oriented compression:

Integers are coded in units of bytes. A variable number of bytes are used to encode a single integer. Variable-Byte, discussed in Section 2.6.3 is an example of this category.

2- Per-integer bit-oriented compression:

This kind of compression schemes uses a variable number of bits to encode a single integer. Golomb-Rice coding, discussed in Section 2.6.4 is an ideal example of this kind of coding. Bit oriented compression schemes result in low decompression speed since bit by bit look up imposes a high processing burden on computer architecture [9].

3 - Block-based compression:

These schemes use a fixed number of input integers and output a variable number of bytes. Frame Of Reference and PForDelta are examples of this kind of scheme. These schemes are discussed in Sections 2.6.6.

Fixed Length Output:

In this compression scheme, each step takes a variable number of integers and produces a compressed form of those integers using a fixed number of bits as a unit. The basic strategy is to pack as many integers as possible from input into a fixed length codeword. The number of input sequences can be identified by using a flag which can be placed in front of every codeword. Simple9 (S9) coding is example discussed in Sections 2.6.5.

We will briefly discuss a few of the compression techniques that founded useful in order to enhance the compression ratio and performance of information retrieval in the following sections.

2.6.1- Block Coding:

Data of any posting list is typically organized in blocks consisting of a few rows every block will contain a number of rows. It is also called a page, though some authors might consider a page to be larger than a block. The size of the block varies according to the policy used in the disk. Compression allows us to store more rows in a single block. Block size may influence the compression since large block size may have negative effects like high memory usage and more time for decompression.

2.6.2- Difference Coding:

This coding mechanism (sometimes called deltas,gaps) [6]encodes a set of integers by deducting successive values. This sort of coding has been used in the field of data compression for years.

The main idea is to code the first value as it is and the remaining values will be coded as the difference between successive values. It achieves a good compression ratio if the differences between successive values are small.

Delta coding is one of the popular techniques for integer coding. It codes integer values by subtracting successive values using simple arithmetic,$\delta = Xi - (Xi - 1)$. The integers must be sorted. If the difference between two consecutive integers is small then we can code the difference with few bits, but one large difference can adversely affect the total compression ratio. Consider a sequence of integers like 24, 25, 32, 43, 55, 77. According to difference, we will store the first value as it is 24. I will store the differences of successive values rather than storing the original values. As a result, the coded sequence will be 24, 1, 7, 11, 12, 22. I can revert back to the original sequence by computing the prefix sum.

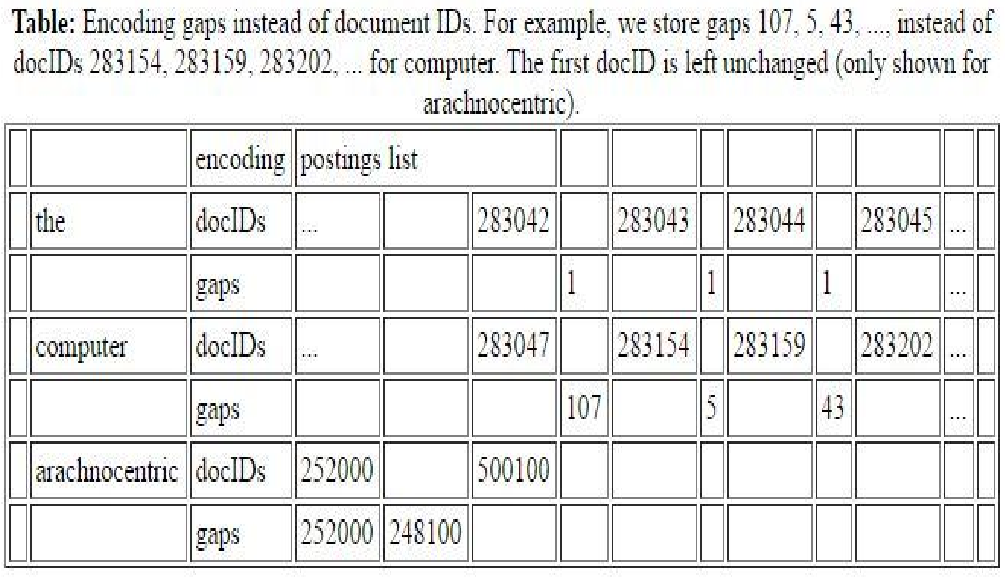


Table: Encoding gaps instead of document IDs. For example, we store gaps 107, 5, 43, ..., instead of docIDs 283154, 283159, 283202, ... for computer. The first docID is left unchanged (only shown for arachnocentric).

		encoding	postings list										
	the	docIDs	...		283042		283043		283044		283045	...	
		gaps				1		1		1		...	
	computer	docIDs	...		283047		283154		283159		283202	...	
		gaps				107		5		43		...	
	arachnocentric	docIDs	252000		500100								
		gaps	252000	248100									

Fig 2.1 Difference Coding (Deltas)

2.6.3- Variable-Byte Coding

An integer can be compressed as a sequence of bytes using Variable-Byte coding. Different authors referred it variously such as v-byte, variable byte [10], VB [11]. This is a byte oriented coding technique. For every byte, the first 7 bits are used to store part of the binary representation of the integer, and the last bit is used for a status bit which indicates whether the next byte is part of this integer [12]. For example, an integer k = 312 and its binary representation is 100111000. Using variable byte coding k can be represented by two bytes: **1**0000010 **0**0111000. The status bit 0 in the first byte (**0**0111000) indicates that the next byte is also part of the integer [17]. (More in Chapter 3).

2.6.4- Golomb-Rice Coding

In Golomb coding, we code an integer (i), by the quotient (q) and remainder (r) of division by the divisor (d). We write the quotient bi=dc in unary notation and the reminder i mod d in binary notation [12]. We need a stop bit after the quotient. We can use 1 as stop bit if the quotient is written as 0 to represent the unary form. In case of Rice coding, we use the divisor as a power of 2. For example if we are coding a number 15 with divisor 4, the code will be 000111. Golomb-Rice coding will achieve good compression ratio. However, the decompression speed is quite slow due to the leading unary values as we need to check a single bit at a time during decompression [13].

2.6.5- Simple9 (S9) Coding

The basic idea of this coding technique is to pack as many integers as possible within 32 bit words. Simple9 divides each block into 4 status bits and 28 data bits. These 28 bits actually store the binary representation on input blocks. These data bits can be divided in 9 different ways: 28 1-bit numbers, 14 2-bit numbers,7 4-bit numbers, 4 7-bit numbers, 2 14-bit numbers, 1 28-bit number, 9 3-bit numbers(1 bit unused), 5 5-bit numbers(3 bits unused), 3 9-bit numbers(1 bit unused). The 4 status bits of a 32-bit word stores which of the 9 cases is used. During decompression, switch cases will be used on status bits to identify each of the 9 cases and apply a bit-mask to exact the desired bits from 32 bit word. Thus, we can recover the original integers [13].

2.6.6-Frame of Reference

Frame Of Reference (FOR) maps a set of large integers to relatively small integers. The basic approach is to identify the range (maximum m and minimum value M) from the set of integers. The smallest number will be coded as 0 and we will store the difference between the smallest number and the original number rather than storing the original set [14].

Each integer can be stored using [log2 (M - m + 1] bits. Alternatively, if the integers are small, we may simply store the number of bits [log2 (M – m + 1] required to store each integer: we call this variation Binary Packing. Different authors have called FOR by different names: Anh and Moffat called it PackedBinary [13].

The first step of implementing this scheme is to partition the array of values into blocks (e.g., 128 integers). We have to compute the range of any particular partition and code the range accordingly. Afterwards, all values in a certain block are coded in reference to the range values. Consider a sequence of integers: 67, 78, 85, 96, 98. We can find that the numbers range from 67 to 98. Using FOR, instead of storing the original sequence we can subtract 67 from each of the values and code the difference only. It transforms the sequence into 0, 11, 18, 29, 31. We can now code each of the offset values using a maximum of 5 bits. We still need to store the minimum value 67 in full and we need 3 bits to store the fact that 5 bits have been used to store differences.

The main drawback of this approach is that a single outlier value might have a significant adverse effect on compression.

2.7- Related Works:

Some work has been proposed to compression of inverted index in IR. To compression the inverted index the researchers have used different compression techniques. We show some of this that related with compression inverted index.

(Matteo Catena, Craig Macdonald, and Iadh Ounis, 2014) [7] They show the little recent work in the literature that thoroughly compares and analyses the performance of modern integer compression schemes across different types of posting information (document ids, frequencies, positions).they experiment with different modern integer compression algorithms, Variable byte codec [10] is a byte-aligned, oblivious codec. It uses the 7 lower bits of any byte to store a partial binary representation of the integer x. It then marks the highest bit as 0 if another byte is needed to complete the representation or as 1 if the representation is complete. For example, 201 is 10000001 01001001. Their finding while this codec may lead to larger representations, it is usually faster than Gamma in term of decompression speed.

(Veluchamy Glory, Sandanam Domnic, 2014) [15] Also carried out an extensive experimental evaluation on synthetic data.They have studied and analyzed various compression techniques for 32-bit integer sequence. They propose a new compression technique called Optimal FastPFOR, based on FastPFOR. FastPFOR technique is a recent fast decoding technique, but compression performance is not as high compared to OptPFD technique .OptPFD technique give the better compression rate, but it does not decompress fast. They technique achieves better compression performance and decompression performance compared to FastPFOR, OptPFD and other techniques. They found that Variable-Byte compared to bitwise technique like Rice code, Variable Byte code required a single branching condition for each byte which is more cost-effective in terms of CPU cycles. Moreover, VByte has a poor compression ratio since it requires one full byte to encode small integers.

(Renaud Delbru, Stephane Campinas, Krystian Samp, Giovanni Tummarello, 2010) [16] They introduce a new class of compression techniques for inverted indexes, the Adaptive Frame of Reference (AFOR) that provides fast query response time, good compression ratio and also fast indexing time. They show that significant performance improvements can be achieved.

AFOR is specifically designed to increase update and query throughput of web search engines. They compare AFOR to alternative approaches, and show experimental evidences that AFOR provides well balanced performance over three factors: indexing time, querying time and compression ratio. In addition, AFOR is simple to implement and could become a new compression method of reference.

AFOR is a straightforward extension of FOR, and they admit that the technique is rather simple. However, they stress that this was the design requirement for our extension since we believe that simplicity over complexity is crucial for achieving high performance in compression algorithms.

2.8- summary:

In this chapter we describe the basic concepts that are required to conduct this research and related work.

In next chapter we explain the old scheme of compression (Variable-Byte) and introduce and explain to enhance scheme (Variable-5bits).

ChAPTER THREE

ANALYSIS AND DESIGN

3- Analysis and Design

3.1- Introduction:

As knew that this research was based on the idea of integers compression. In this section we will explain the old method of compression algorithm and then show weaknesses. After this we will explain the new algorithm (improving the old algorithm).

3.2- Previous Scheme:

Variable-Byte Coding:

An integer can be compressed as a sequence of bytes using Variable-Byte coding. Different authors referred it variously such as v-byte, variable byte [10]. This is a byte oriented coding technique. For every byte, the first 7 bits are used to store part of the binary representation of the integer, and the last bit is used for a status bit which indicates whether the next byte is part of this integer [12]. For example, an integer k = 312 and its binary representation is 100111000. Using variable byte coding k can be represented by two bytes: **1**0000010 **0**0111000. The status bit 0 in the first byte (**0**0111000) indicates that the next byte is also part of the integer [17] **See Fig 3.1.**

In total, it took 16 bits to encode the number 312. While decoding, we read byte by byte. We discard the eighth bit if it is 0 and continue reading the next bytes till we get 1 in the eighth bit for any byte to record any integer. Variable-Byte coding is easy to implement and known to be significantly faster than traditional bit oriented methods [12].

The main drawback of this method is that it will take at least 1 byte to store even a small integer. In the case of encoding an integer of a single bit such as 1, we have to store it using one byte and the byte looks like 10000001. As a result, we waste 7 bits. Thus, it can make Variable-Byte coding inefficient in terms of compression ratio.

docIDs	200	205	214782
gaps	200	5	214577
VB code	**1**000001 **0**1001000	**1**0000101	**1**0001101 **0**001100 **0**0110001

Fig 3.1 Variable-Byte Scheme

3.3- Enhanced Scheme (Variable-5bits):

Namely enhanced scheme, we proposed scheme which we called **V5bits** it enhanced to VByte scheme. An integer can be compressed as a sequence of 5 bits using Variable-5bits coding. This is a 5bits oriented coding technique. For every 5bits, the first 4 bits are used to store part of the binary representation of the integer, and the last bit is used for a status bit which indicates whether the next byte is part of this integer. For example, an integer k = 200 and its binary representation is 11001000. Using V5bits coding k can be represented by 10 bits: **1**1100**0**1000. The status bit 0 in the first 5 bits (**0**1000) indicates that the next 5 bits is also part of the integer.

In total, it took 10 bits to encode the number 200. While decoding, we read 5 bits by 5 bits. We discard the 5 bits if it is 0 and continue reading the next 5 bits till we get 1 in the 5 bit for any

5 bits to record any integer **See Fig 3.2**. Variable-5bits coding is easy to implement.

docIDs	200	205	214782
gaps	200	5	214577
VB code	**1**1100 **0**1000	**1**0101	**1**0011 **0**0100 **0**0110 **0**0011 **0**0001

Fig 3.2 V5bits

To quantify the trade-offs among between two algorithms, we have to implement apply them to randomly generated data sets. We use some code for compression and decompression of integers written in Java which is available as open source under the Apache License 2.0 (https://github.com/lemire/JavaFastPFOR). Along with the implementation of the core compression algorithm, we need to develop I own testing strategy to test and verify. All the implementations have been written in Java using SDK version 1.8.0

In our experiment we focus on compression of integer values. The main objective of this study is to examine and compare the performance of enhanced V5bits scheme over VByte method. Mainly, we comparing between them based on compression ratio and decompression speed. We conducted our experiment based on synthetic data sets which will be discussed in Chapter 4. We can reproduce synthetic data at any time and perform testing on those data. We run different trials by varying different parameters such as number of integers, integer size, etc.

In order to run all of our experiments on synthetic datasets we use a machine equipped with Intel(R) Pentium(R) CPU B960 @ 2.20 GHz 2.20 GHz processor and 2 GB of RAM.

3.4- Summary:

In this chapter we explain the variable-byte algorithm and then show weaknesses. After this we will explain the variable-5bits algorithm. In the next chapter, we discuss experiment on synthetic data. We describe the nature of synthetic dataset and discussion and assess the results.

CHAPTER FOUR

EXPERIMENT AND DISCUSSION AND EVALUATION

4.1-Synthetic Data

To evaluate the performance of two compression schemes we performed thorough experiments on synthetic data. We varied different parameters such as integer size, number of integers, etc. We analyzed the effect of different aspects of datasets on compression.

Again, size of integers may influence the performance of compression. Will take more bits to compress one large integer.

We can save space by compressing small integers into smaller memory space compared to larger integers. Moreover, the difference between the consecutive integers also influences the compression ratio when we use delta coding or differential coding. The cardinality of the data may influence the compression rate.

We have identified different factors that can influence the compression size. Our final goal is to examine and compare the performance of two compression schemes on integer’s numbers. In Our experiment we examined not only compressed size but also compression and decompression speed. We examined how different factors can influence the time to compress and decompress data.

To evaluate the performance we ran several identical trials varying characteristics such as the number of integers and the maximum length of each integer. In the case of delta coding, the cardinality of data can influence the compression speed. In the case of low cardinality and sorted sequence, delta values will be smaller. So, we can examine the compression speed by varying the cardinality of data.

In this chapter, we discuss the generation of our synthetic data. Afterwards, we describe our experiment varying different characteristics of data. Finally, we analyze and discussion the result.

4.2 Generation of Data and Discussion the Result:

To perform the experiment, we generated data with two different parameters.

- Fixed number of integers
- Variable number of integers

After generate integers by two different parameters we applied concept of Delta coding or differential coding between integers on two parameters.

However, these synthetic tests provide us with a reference. We can only measure the performance of two compression algorithm based on the comparison characteristics (See Section **2.5**).

As discussed in the previous section we varied different characteristics of data. Therefore, our first approach was to generate generating consecutive integers from zero to a certain limit. We gave two parameters N and Max. Our program generated consecutive integers N distinct integers from 0 to Max and stored it to an array of integers in sorted order. We used differential coding (See Section **2.6.2**) on the input values and stored the delta values in an array of integers. This set of delta values was used as an input to evaluate different between two compression schemes.

In our implementation we took a delta coded integer array as an input and ran two compression schemes and recorded the compression ratio, compression speed and decompressing speed for every scheme.

In the case of differential coding, we storing the difference between successive integers together with initial value: (x1, x2 = x2 - x1, x3 = x3 – x2 , ….) instead of storing the original array of integers in sorted order such as x1, x2, … where xi+1 >= xi. This will result in non-negative integers that are typically much smaller than the original integers.

Thus, we will be getting benefit for compressing a sequence of smaller numbers rather than the original one.

In all of our experiments, we measured the timing based on CPU time. In the case of compression and decompression, we ran every trial for 10 times and took the average of the recorded time for every trial. While measuring the timing we consider all the encoding and decoding operations including delta coding and prefix sum.

4.2.1 Varying Integer Size

We generated our synthetic data sets with random integers in the range [0 – 2^^{31}) for both V5bits and VByte schemes. We generated integers from zero to the integer with the specified number of bits, i.e., the range will be from 0 to a maximum 31 bit integer which means the interval is from 0 to 2147483648. Our test includes the integer range [0 – 2^^{16}] [0 – 2^^{17}], [0 – 2^^{18}) and so on. We generated 32768 numbers of distinct integers for every interval in sorted order and stored them into an array for further processing. Finally, we stored the delta values. We will get larger delta values with increase of the range, since us generating fixed number (32768) of distinct integers in a particular interval. We analyze the effect of integer size on compression. If we have all very small integers we can fit those into small memory space. We have listed the test result for both V5bits and VByte schemes. In our experiment with synthetic data

Since us trying to evaluate the performance of two compression schemes, we tried to measure the compression ratio, compression speed and decompression speed for two compression schemes.

We have plotted the test results in Fig. **4.1**. , Fig. **4.1** shows the result of V5bits and VByte scheme. We found the V5bits is more compressible compared to VByte in small integers and equal in large integers. In this test we changed the size of integer to some certain limit. When we generate integers within small range such as [0 – 2^^{16}], it is obvious that we will get small delta values compared to larger range like [0 – 2^^{31}]. Therefore, when we have lot of small integers, we can compress them into smaller memory space. For both V5bits and VByte data in Fig. **4.1**, we found small integers can be compressed into fewer bits compared to large numbers. In the case of maximum 16 bit integers, we can compress every integer within 5 bits or 10 bits there is an exception for Variable-Byte, since it compresses in a byte oriented manner. Therefore, Variable-Byte needs at least 8 bit to compress any integer.

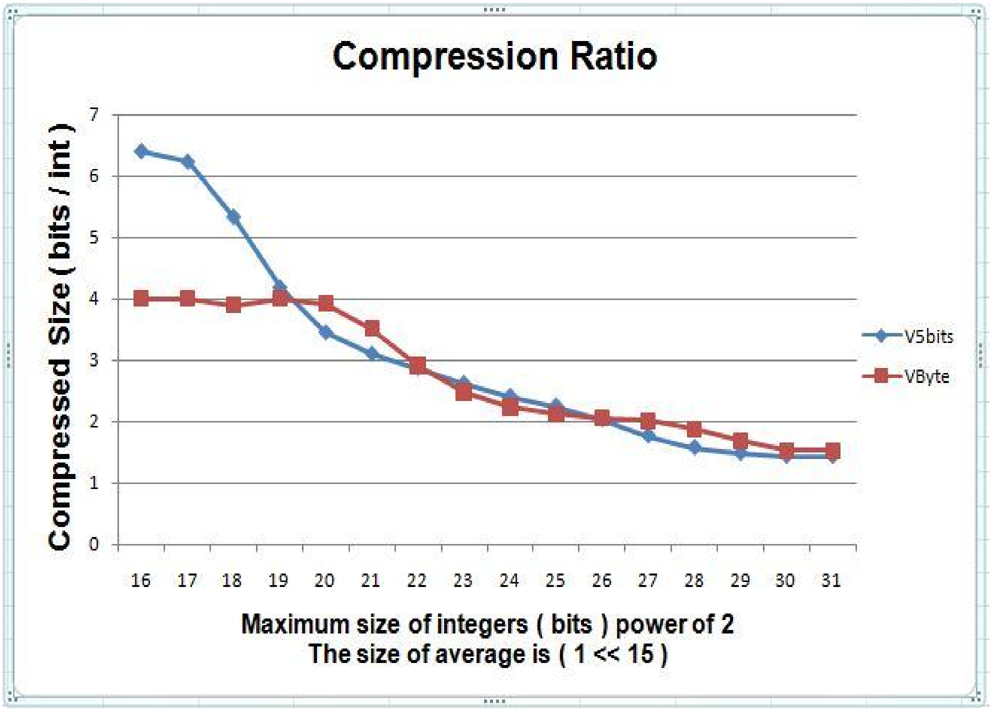


Figure 4.1 Compression Ratio

Size of integers has some effect on compression speed as well. Compression speed decreases with increase of size of integer. In Fig. **4.2**, we found compression speed is higher for small integers. Compression speed decreases with increment of integers size. We measure our compression speed in million of integers per second (mls). The V5bits scheme has the lowest compressing speed compared to VByte, because the V5bits after convert the integer into binary reading 4 bits per reading compared with VByte reading 8 bits. To evaluate the applicability of using any compression algorithm, we are mostly concerned about the compressed size and decompression speed of the scheme. As a result, most of the algorithms we used have lower compression speed compared to decompression speed.

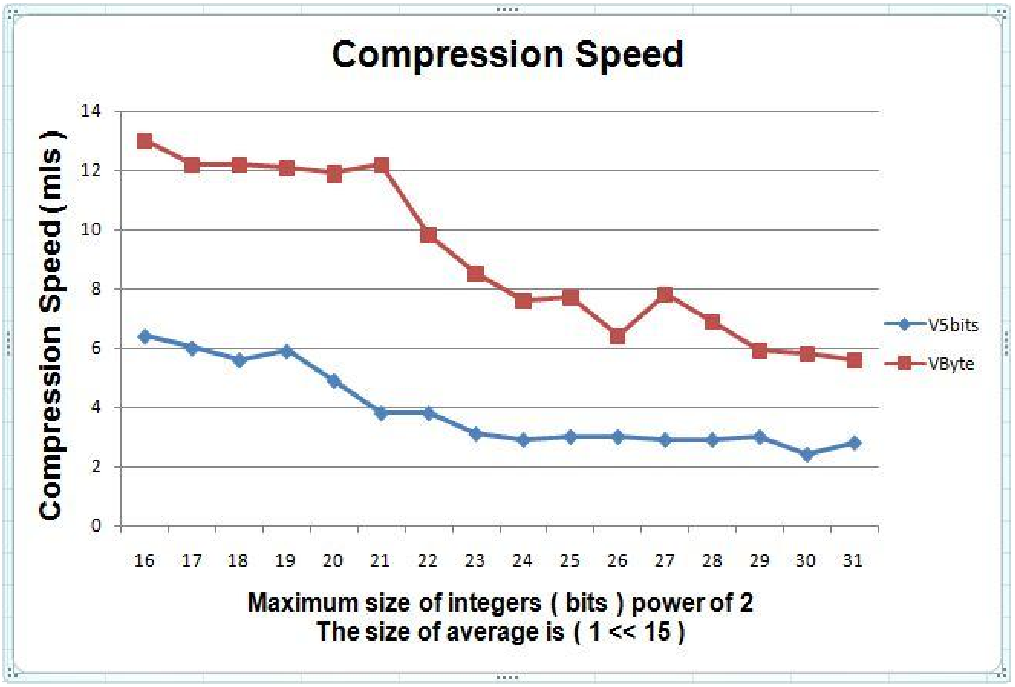


Figure 4.2 Compression Speed

Decompression speed was also influenced by integer size see Fig **4.3**. We used the same number of integers in every trial. We found the integer size influenced the performance of compression in every trial. In the case of the first trial, we have maximum number of bits is 16. Therefore, the range of random generated integer is in between [0 – $2^{\wedge 16}$]. The difference between consecutive integers is small for a small range of integers.

In the case of a large range such as [0 – 2^31], the difference between consecutive integers is large and we found it takes more bits to compress. We can compress the integers into a smaller number of bits when we have small integers and also compression and decompression time will be significantly high for small integers compared to large integers.

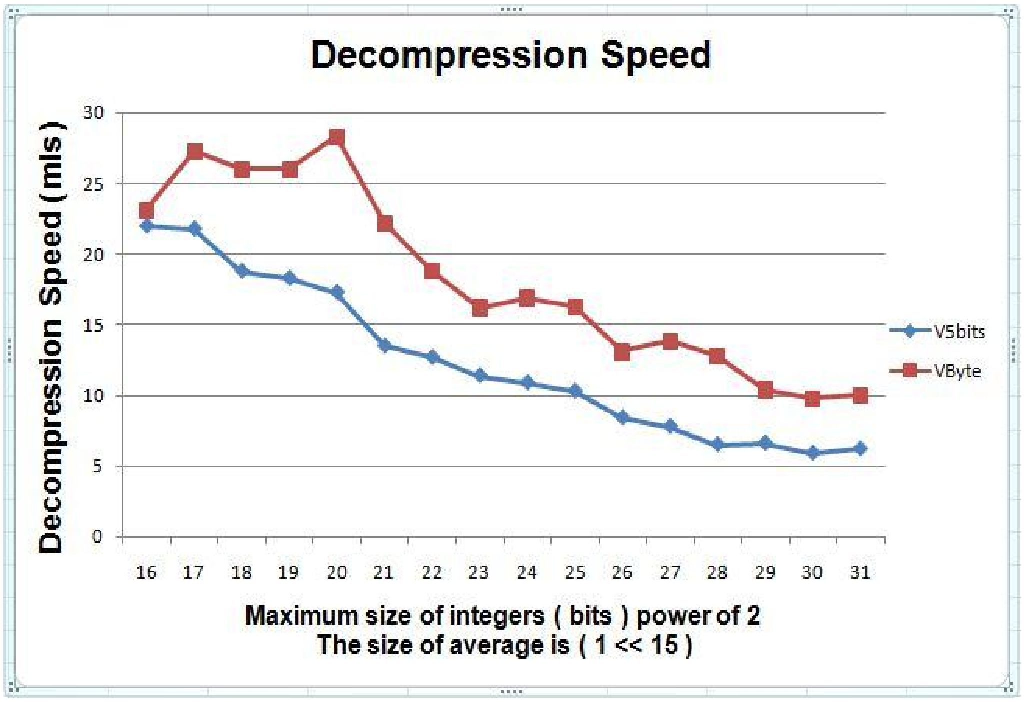


Figure 4.3 Decompression Speed

4.2.2 Varying the Number of Integers

Our next approach to testing was to vary the number of integers within the same (total) range of integers (0 – 2^31) and evaluate the performance. In this case, we generated data sets of random integers for both V5bits and VByte. Rather than generating a fixed number of integers like 32768 we generated a variable number of integers in each run. In every trial, we generated 10 integer arrays. Each of them containing same number of integers in a certain pass and in every pass we changed the size of the array by changing the number of input integers. In our experiments we generated random integers within the same range of (0 – $2^{\wedge 31}$) but varying the number of integers i.e., we generated $2^{\wedge 11}$ integers for each input array in the first pass. We continued our trial by generating more integers than in the previous trial by a power of 2. We generated $2^{\wedge 12}$ to $2^{\wedge 22}$ integers in our test. Since the range is fixed, the average difference among random generated numbers will be higher when we generate a small number of integers and reverse in the case of generate a higher number of integers.

In our test, we generated data for both V5bits and VByte. We record average of compression size, compression speed and decompression speed. In this scenario, we ran the first 6 integers identical trials and generated $2^{\wedge 11}$, $2^{\wedge 12}$, … $2^{\wedge 16}$ integers. In every trial, we generated random numbers in the range (0 – $2^{\wedge 31}$). The difference among consecutive numbers is large, since we generating fewer of integers on a fixed large range (0 – $2^{\wedge 31}$). The differences between consecutive integers are known as delta values. In Fig.**4.4** we found that to compress integers with large delta values takes more bits. Moreover, it takes more time to decompress.

In the case of ran the last 6 integers, we generated $2^{\wedge 17}$, $2^{\wedge 18}$, … $2^{\wedge 22}$ integers in the same range (0 – $2^{\wedge 31}$). Delta values will be lower in high integers. As a result, we can achieve very good compression ratio and also good decompression speed.

In Fig. **4.4**, we can see the effect of number of integers on compression size to both V5bits and VByte schemes. Compression size is increasing whenever we generating more integers on a fixed range (0 – $2^{\wedge 31}$). We will get small delta values when we have more integers. As a result we can compress integers with fewer bits.

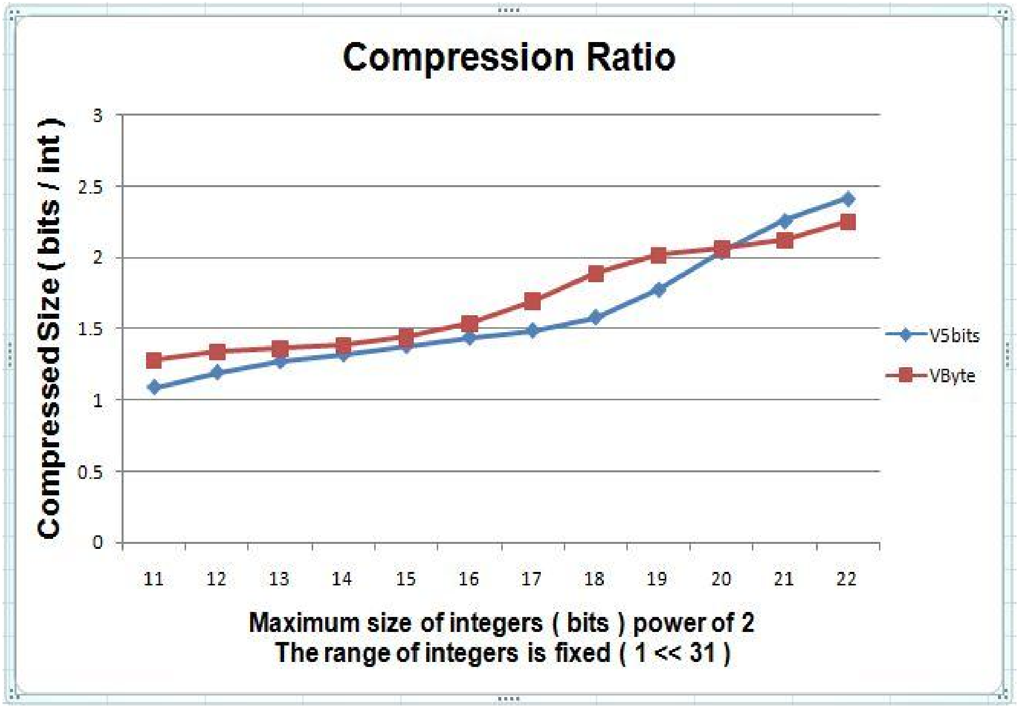


Figure 4.4 Compression Ratio

Fig. 4.5 shows the compression speed. We Compression speed increases with the increment of number of integers, since delta value decreases as the number of integers increases. We can see the effect of the number of integers on compression speed in Fig.**4.5**. VByte compression speed faster than V5bits.

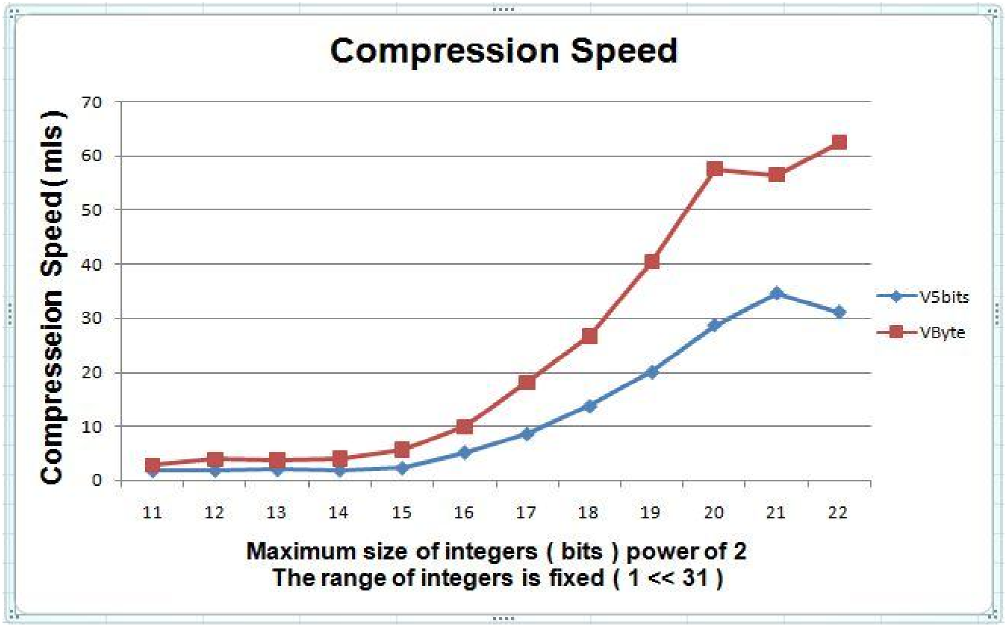


Figure 4.5 Compression Speed

We can see in Fig. **4.6**, decompression speed increases with the increase in number of integers.

VByte faster in terms of decompression speed than V5bits.

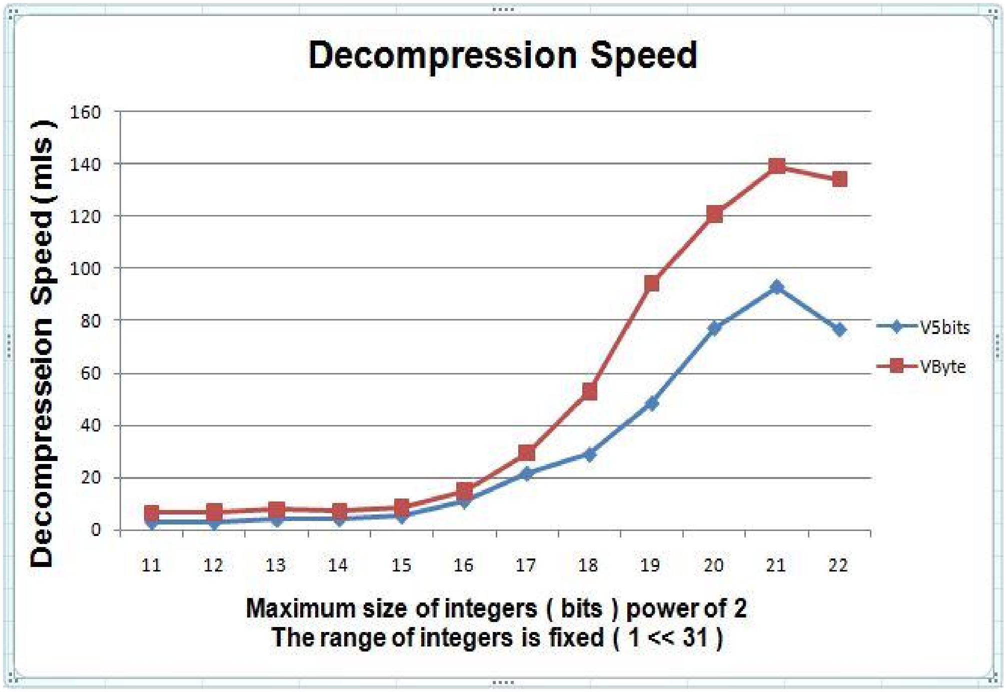


Figure 4.6 Decompression Speed

In our synthetic data test, we generated a fixed number of integers and also a variable number of integers, but in both of our cases we have seen the size of integers influences the compression ratio and speed of compression and decompression.
However, V5bits is clearly the winner for compression size when generated small integers in case of fixed number of integers, and we generated more integers in case of variable number of integers.
VByte is clearly the winner for speed of compression and decompression but it does not compress.

4.3- Summary:

In this chapter we generating synthetic dataset and worked it for show the results and discussion and evaluation this results.
Next chapter include conclusion and future work.

CHAPTER FIVE

CONCLUSIONS AND FUTURE WORK

5- Conclusions and Future Work

In this Chapter, summarized our work and review our experimental results and the contribution of our study. Moreover, explore the possible extension of our work.

Interested in measuring the performance of variable byte integer compression over randomly indexes. We main criteria to evaluate compression scheme is compression ratio, compression speed and decompression speed. We mostly concerned about the compression ratio and decompression speed.

Contributions

Enhanced Variable-5bits algorithm provides the best compression ratio for small integers compare with Variable-Byte scheme in case of fixed number of integers, and we generated more integers in case of variable number of integers in addition to introduce trade-off between compression ratio, compression speed and decompression speed.

Future Work

The performance of Variable-5bits compression schemes tested in the case of synthetic data. In future we can evaluate the outcome of these schemes by real dataset to assess real performance. Observe in the results of experiments that did not get a high compression ratio for large numbers, we will work to improving compression ratio for large numbers.

References:

[1] Kobayashi, M. and Takeda, K.:Information retrieval on the web, ACM Computing Surveys, Vol.2, No.2, pp.144-173(2000)

[2] Dean, J.: Challenges in building large-scale information retrieval systems: invited talk. In: Proc. WSDM ’09. (2009)

[3] Anh, V.N. and Moffat, A.: Inverted index compression using word-aligned binary codes, Information Retrieval, Vol.8, No.1, pp.151-166 (2005)

[4] Tortman, A.: Compressing inverted files, Information Retrieval, Vol.6, No.1, pp.5-19 (2003)

[5] Buttcher S, Clarke CLA. Index compression is good, especially for random access. Proceedings of the 16th ACM conference on Information and Knowledge Management, CIKM ’07, ACM: New York, NY, USA, 2007; 761–770, doi:10.1145/1321440.1321546.

[6] Witten, I.H., Bell, T.C., Moffat, A.: Managing Gigabytes: Compressing and Indexing Documents and Images. 1st edn. (1994)

[7] On Inverted Index Compression for Search Engine Efficiency, Catena, Matteo, Macdonald, Craig, and Ounis, Iadh – 2014, Publisher: Springer.com.

[8] Manning, C.D., P. Raghavan, and H. Schütze, *Introduction to information retrieval*, Cambridge University press Cambridge, Vol.1. 2008.

[9] F. Scholer, H. Williams, J. Yiannis, and J. Zobel. Compression of inverted indexes for fast query evaluation. In SIGIR’02, pages 222–229, New York, NY, USA, 2002. ACM.

[10] Williams, H.E., Zobel, J.: Compressing integers for fast file access. The Computer Journal 42 (1999).

[11] A. Stepanov, A. Gangolli, D. Rose, R. Ernst, and P. Oberoi. Simd-based decoding of posting lists. In Proceedings of the 20th ACM International Conference on Information and Knowledge Management, CIKM ’11, pages 317–326, New York, NY, USA, 2011. ACM.

[12] J. Zhang, X. Long, and T. Suel. Performance of compressed inverted list caching in search engines. In WWW ’08, pages 387–396, 2008.

[13] V. N. Anh and A. Moffat. Inverted index compression using word-aligned binary codes. Information Retrieval, 8(1):151–166, 2005.

[14] J. Goldstein, R. Ramakrishnan, and U. Shaft. Compressing relations and indexes. In ICDE ’98, pages 370–379, Washington, DC, USA, 1998. IEEE Computer Society.

[15] Compression Inverted Index Using Optimal FastPFOR V Glory, S Domnic - Journal of information processing, 2014 jlc.jst.go.jp.

[16] Adaptive Frame Of Reference For Compressing Inverted Lists. Renaud Delbru, Stephane Campinas, Krystian Samp, Giovanni Tummarello. DERI TECHNICAL REPORT 2010-12-16 DECEMBER 2010.

[17] Performance Evaluation of Fast Integer Compression Techniques Over Tables, Ikhtear Md. Sharif Bhuyan, Dean of Graduate Studies, The University of New Brunswick, November 2013.